NEURON® CHIP Performance Using the Dhrystone C Benchmark

August 1991 **LONWORKS™ Engineering Bulletin**

Introduction

The Echelon LONBUILDER™ Developer's Workbench for the NEURON CHIP includes a cross-compiler for the NEURON C language. NEURON C is based on the ANSI C language standard with extensions for multi-tasking, embedded sense and control, and networking. This application note describes what is involved in porting a typical C benchmark program to NEURON C, together with the results obtained from compiling and executing the benchmark.

In this case, the program was Reinhold Weicker's Dhrystone C/1.1 benchmark (see Communications of the ACM, vol. 27). This program is an integer program that measures the performance of a processor/compiler combination. It is written in a pre-ANSI dialect of C known as K&R C. The benchmark was ported in order to do the following:

- Describe the small changes needed to an ANSI C program in order to compile it as a NEURON C program.
- Benchmark the performance of a NEURON CHIP in comparison with an 8051-type microcontroller.
- Benchmark the object code space efficiency of a given C program in a NEURON CHIP environment in comparison with an 8051-type microcontroller.

The changes required to the Dhrystone benchmark program for it to run on the NEURON CHIP fall into two classes:

1. Changes due to the difference in run-time environments between a typical computer with a C compiler, and a NEURON CHIP running NEURON C.

2. Implementation dependencies of the NEURON C compiler.

Run-Time Environment Changes

1. Data beyond 256 bytes must be declared `far`.

NEURON C recognizes two data segments, `near` and `far`. By default, data is allocated to the `near` segment which is 256 bytes long. If a program declares more

than 256 bytes of static or global data, this must explicitly be allocated to the `far` segment.

2. Some `int`s should be promoted to `long`.

Most C programs assume that the data type `int` is at least 16 bits long. However, the NEURON C `int` data type is only eight bits long. If the program relies on some `int` objects being 16 bits long, then these need to be modified to `long` type. Note that the Dhrystone benchmark measures the performance of `int` arithmetic, which is eight bit arithmetic for NEURON C.

3. Add `watchdog_update()` calls in time-consuming loops.

The NEURON CHIP implements a watchdog timer which will reset the chip unless periodically refreshed. Typical multi-tasking programs relinquish control to the task scheduler frequently enough that the scheduler can take care of this. However, a compute-bound program such as a benchmark may cause a single task to run longer than the watchdog period (which is 0.84 seconds at 10 MHz input clock). Therefore the `watchdog_update()` system call should be used periodically to avoid the automatic reset.

4. Add `when(reset)` task to call `main()`.

The standard C environment assumes a single-threaded program invoked by calling the special entry point `main`. However, a NEURON C program consists of multiple independent tasks activated by a scheduler, and there is no main entry point. The effect of a standard C environment can be simulated by creating a task that is activated when the chip is reset, and this task then calls `main`. See the appendix for an example of this code. Note that there is no mechanism in the NEURON C environment to pass command line arguments to `main` with the `argc, argv` parameters.

Implementation Dependencies

1. Declare arrays, `structs` and `unions` local to a procedure as `static`, and replace array and `struct` initializers with assignment statements.

NEURON C does not support `auto` (stack resident) arrays, `structs` or `unions`. A simple workaround is to declare these as `static` (data segment resident). However, if the procedure declaring the object is re-entrant, this will not work correctly. Note that the data stack of the NEURON CHIP is fixed in size (data stack size + return stack size < 256 bytes), so that large amounts of data cannot be stored locally. Also, initializers for aggregates are not provided. You can replace them

with assignment statements at the beginning of the function (for `autos`) or at the beginning of `main` (for `statics` or globals).

2. Comment out `#ifdef .. #else .. #endif` appropriately, and expand parameterized macros.

The NEURON C pre-processor does not support conditional compilation or parameterized pre-processor macros. The workaround is to expand these manually before compiling.

3. Add `return` statements at the end of functions that do not already have one.

NEURON C will fail to compile a non-`void` function that does not have a `return` statement as the last syntactic statement, even though the end of the function cannot be reached.

4. Remove calls to `printf` – use debugger breakpoints instead.

NEURON C does not have a complete ANSI C run-time library. Many of these functions are inappropriate for a target machine that has no standard I/O device or file system. In the case of the `printf` function, which performs formatted I/O to the standard output device, a simple workaround is to use the capability of the LONBUILDER's symbolic debugger to display formatted data. Using the *eval* command of the debugger allows you to examine the value of program variables on the screen of the LONBUILDER's host computer. The debugger also has the ability to retrieve the error log from the NEURON CHIP, and display a listing of errors detected by the NEURON CHIP's run-time system.

5. Implement `malloc, strcpy, time, exit, strcmp` etc.

There are other, non-I/O related, functions in the ANSI C library that are not in the NEURON C library. See the appendix for NEURON C code that implements those functions necessary to run the benchmark.

6. Add prototypes for all functions that are forward-referenced.

NEURON C is stricter than ANSI C with respect to forward-referenced functions. NEURON C requires a function prototype to be defined before any forward reference to that function is made, whereas ANSI C does not. Simply define function prototypes for all functions in the source file before any calls are made.

7. `Boolean, TRUE` and `FALSE` are already defined by the compiler.

NEURON C predefines the data type `Boolean`, and the literals `TRUE` and `FALSE`. ANSI C does not predefine them, but they are commonly defined by the user's source code. If the program contains such definitions, they should be removed to avoid a doubly-defined symbol compilation error.

Results

The Dhrystone benchmark was successfully ported to the NEURON CHIP with the changes described above. The results of the benchmark are that the NEURON CHIP with a 10 MHz input clock and the NEURON C version 1.0 compiler executes 108 Dhrystones per second. The performance scales linearly with clock speed. The total performance of the NEURON CHIP is three times this value, since it contains three identical CPUs. The Dhrystone benchmark itself executes on one of the three CPUs; the other two are used for LONTALK™ network protocol processing. Both the NEURON 3120™ CHIP and the NEURON 3150™ CHIP have identical performance.

The source code is 195 lines, and the object code is 1,378 bytes, giving an expansion of seven bytes of object per line of source. The number of lines is counted by counting semicolons in the source code – this approximates the number of executable lines, not including comments and blank lines. The maximum free space for user code in a NEURON 3120 CHIP EEPROM is 420 bytes, which is about a 60 line program. The NEURON 3150 CHIP has 43,008 bytes free space for user code, which translates to a 6,000 line program.

To evaluate the NEURON CHIP with respect to other 8-bit microprocessors, the table presents a comparison to the Intel 8051 processor and Franklin Software's C-51 v3.0 compiler. This processor and compiler combination have a raw performance rating of 203 Dhrystones/sec at 12 MHz input clock. Module code size is the size of the compiled code for the benchmark itself.

Total code size includes the runtime library support code. For the NEURON CHIP, the two numbers are the same, because the runtime library is part of the system architecture.

Processor	NEURON CHIP	Intel 8051	
Compiler	NEURON C	Franklin C-51	Ratio
Input Clock (MHz)	10.0	12.0	0.83 : 1
Internal Clock (MHz)	5.0	6.0	0.83 : 1
CPUs on chip	3	1	3 : 1
Dhrystones/sec/CPU/MHz	10.8	16.9	0.64 : 1
Dhrystones/sec/MHz	32.4	16.9	1.92 : 1
Module code size (bytes)	1,378	2,134	0.65 : 1

Total code size (bytes)	1,378	5,601	0.25 : 1

Table 1: Comparative Performance of the NEURON CHIP with NEURON C

Note that the NEURON CHIP shows a 92% performance improvement over the Intel 8051 running the specified compiler. It also shows a 35% space reduction for the module code size, and a 75% reduction for the total code size.

Appendix – NEURON C code to Support Dhrystone Benchmark

This appendix contains the additional runtime code needed to support the Dhrystone benchmark. The ported benchmark code itself is not listed, but is available from Echelon.

```
// Missing string functions

void strcpy( char *dest, const char *source ) {
                                                    // String copy
     while( *dest++ = *source++ );
}

int strcmp( const char *source, const char *dest ) {
                                                    // String compare
     for( ; *source == *dest; source++, dest++ )
          if( !*source ) return 0;
     return *source - *dest;
}

// Time function

stimer global_timer;
long time( long *x ) {          // Seconds since main was called
     return 30000 - global_timer;
}

// Exit function

#include <control.h>
void exit( int ret_val ) {
     global_timer = 0;
     go_offline( );
}

// Primitive heap manager

#define HEAP_SIZE 100
char heap[ HEAP_SIZE ];         // Dynamic memory pool
char *heap_ptr;
char *end_heap;
```

```
void *malloc( unsigned int req_size ) {
     void *blk_ptr;

     blk_ptr = heap_ptr;        // Just get the next chunk
     heap_ptr += req_size;
     if( heap_ptr > end_heap ) return NULL;
     return blk_ptr;
}

void free( void *blk_ptr ) {    // No storage reclamation!
}

// Reset clause

when( reset ) {
     heap_ptr = heap;           // Initialize heap pointers
     end_heap = heap + HEAP_SIZE;
     global_timer = 30000;
                    // Initialize second count-down timer
     main( );                   // Call user's main program
}
```


Echelon Corporation
4015 Miranda Avenue
Palo Alto, CA 94304
Telephone (415) 855-7400
Fax (415) 856-6153

Echelon Europe Ltd
105 Heath Street
London NW3 6SS
England
Telephone (071) 431-1600
Fax (071) 794-0532
International Telephone + 44 71 431-1600
International Fax + 44 71 794-0532

Echelon Japan K.K.
AIOS Gotanda Building #808
10-7, Higashi-Gotanda 1-chome,
Shinagawa-ku, Tokyo 141, Japan
Telephone (03) 3440-8638
Fax (03) 3440-8639

Part Number 005-0005-01